L8a: WebSockets

Web Engineering

194.161 VU SS24

Jürgen Cito

TU WIEN Informatics

IPA

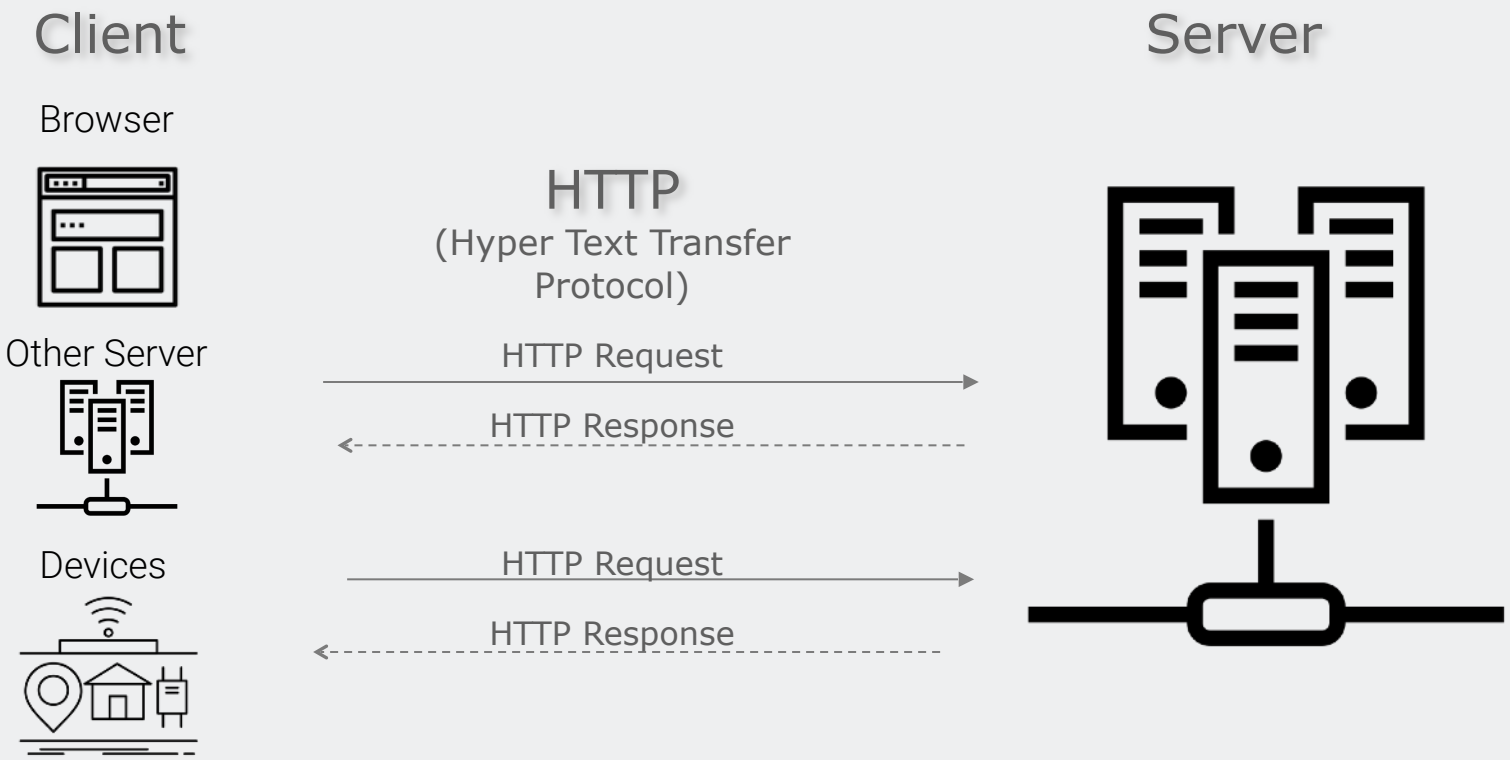
L8a: WebSockets

- Quick recap of HTTP and its relationship to TCP
- Real-time communication on the web
- WebSockets concepts
- Practical instantiation in backends (Node.js) and the browser

Learning Goals

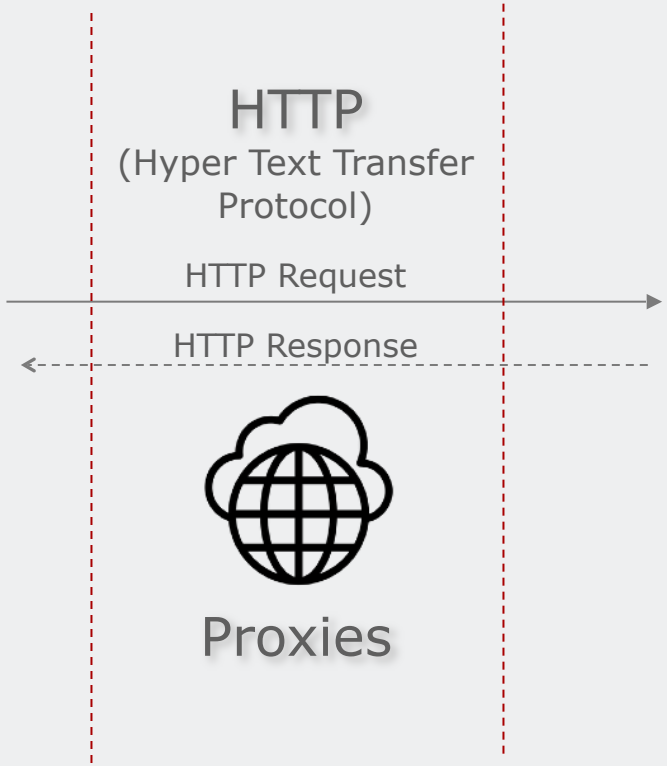
- Describe how clients and web servers interact **bidirectionally**
- Understand the underlying mechanism of how a WebSocket connection is established
- Being able to broadcast messages to different clients on a web socket
- Build a basic WebSockets client and server

Recap: HTTP Request/Response

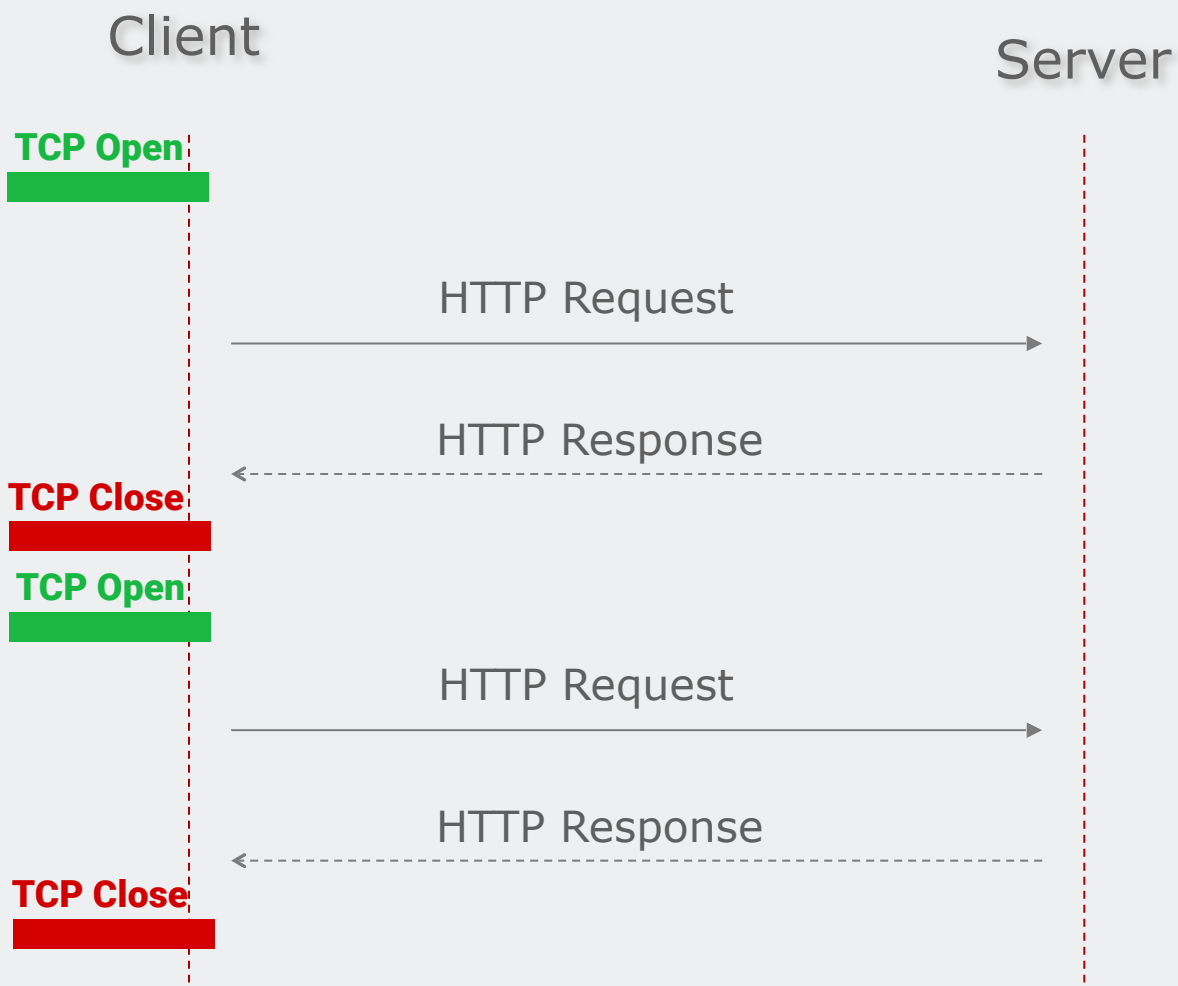


HTTP Overview

- Builds upon TCP/IP
- Synchronous request-response protocol
  - Client (web browser) sends request
  - Web server replies with appropriate answer (could also be an error)
- "Stateless" protocol
  - Each request-response pair is independent
  - No permanent connection between server and browser (allows for a high number of users per server)
- Proxies mediate between browser and server (caching, filtering, etc.)
- In HTTP everything is sent and received as **clear text**
  - **Use HTTPS:** HTTP over a secured (TLS) connection

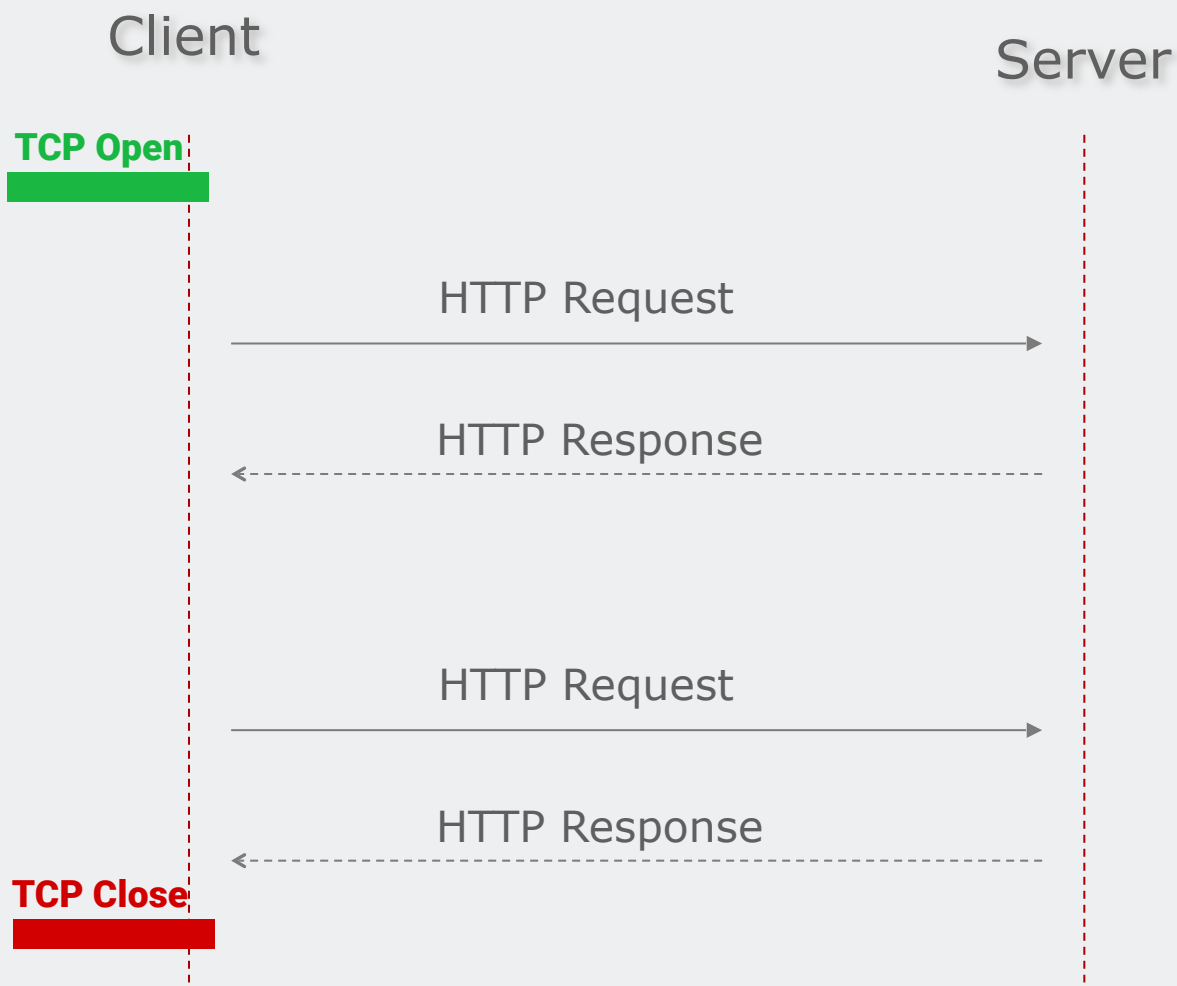


HTTP Overview + TCP Connections
HTTP 1.0 - Multiple TCP connections



HTTP Overview + TCP Connections

Starting with HTTP 1.1: Persistent TCP connections



HTTP 1.0 used the *Connection: keep-alive* **header for persistent connections, which is the default starting with HTTP 1.1**

WebSockets: Real-time communication (1/2)

WebSockets: Communication protocol to send data bi-directionally on the internet

Persistent 2-way connection over a single TCP connection

- Client sends an initial request with an HTTP upgrade header to the server
- If the web server supports web sockets, it replies with an appropriate upgrade response
- From that moment, communication runs on the WebSocket protocol
  —> No HTTP headers (it’s a binary protocol)

WebSockets: Real-time communication (2/2)

Standardized by the IETF in RCF 6455

Stateful protocol

- Permanent connection between server and client
- The server maintains the connection state for each client
  (This leads to increased memory use and potential challenges for scalability)

Use cases

- Chat, Live Leaderboards, Online gaming, Notifications, Collaboration, etc.

WebSockets Clients

WebSockets can also be used with clients that are not browsers

Different clients can interface with WebSocket Servers

- They need to implement the WebSocket protocol
- Often built-in APIs in many languages (e.g., Python)
- Most common client is still web-based in JavaScript

Requires the server to be able to serve WebSockets
(e.g., cannot use WebSockets with REST/HTTP APIs)

“Real-time communication” before WebSockets
Polling & Long Polling

Polling

- Continuously send requests to the server
  (e.g., using *`fetch()`* and setTimin the browser/JS)
- Most requests will probably return an empty response

Long Polling

- Send request to the server, keep connection open until server has new data
- Not recommended for high-traffic scenarios or actual real-time updates

WebSocket URIs

The protocol specification defines two new Uniform Resource Identifier (URI) schemes

- `ws://`
- `wss://` (with encryption using TLS)

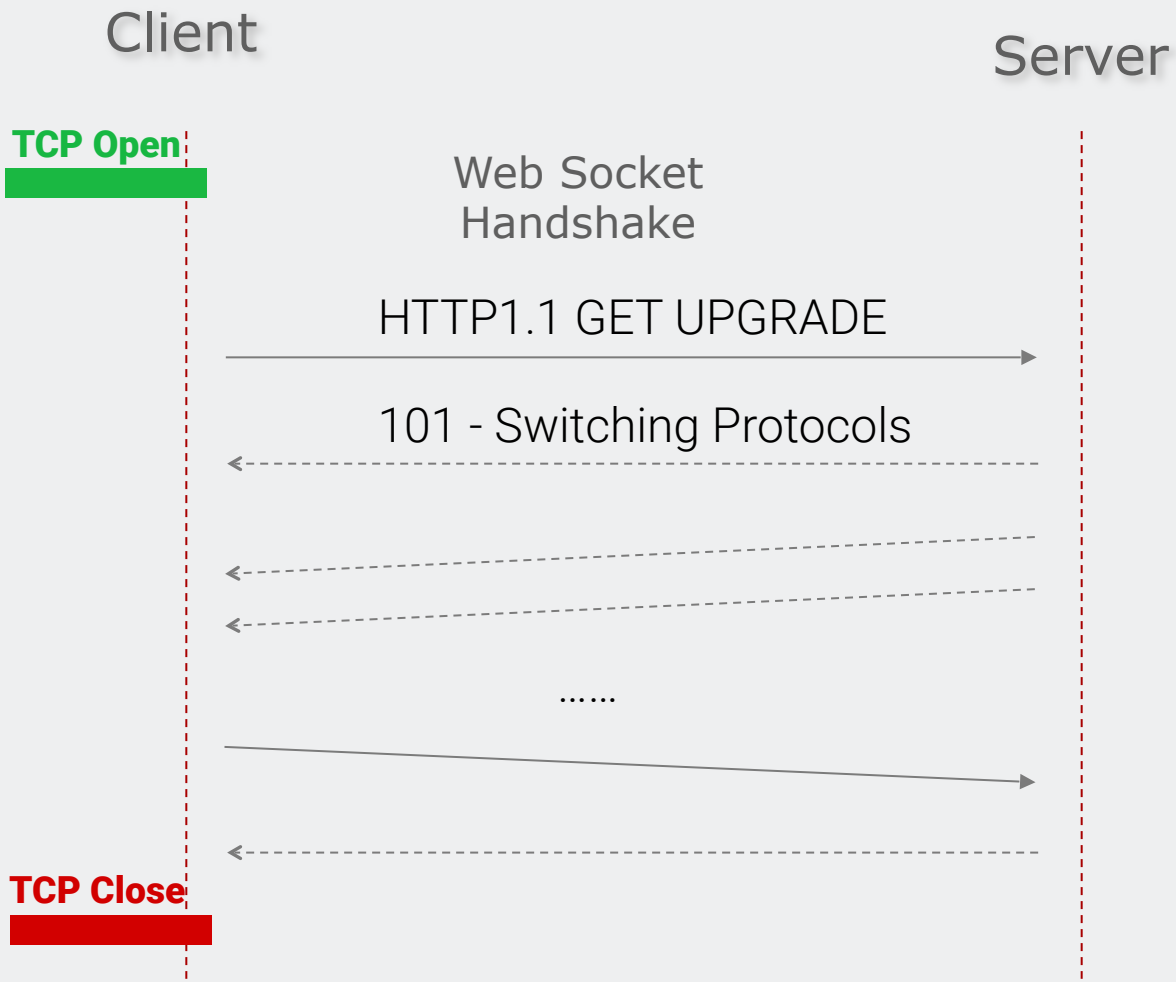
The rest of the URI components

- One exception: WebSocket URIs do not support fragments
  (i.e., no # fragment after path definitions)

Example URI

- wss://www.some.domain/leaderboard

WebSockets Overview



Web Socket Handshake HTTP Headers

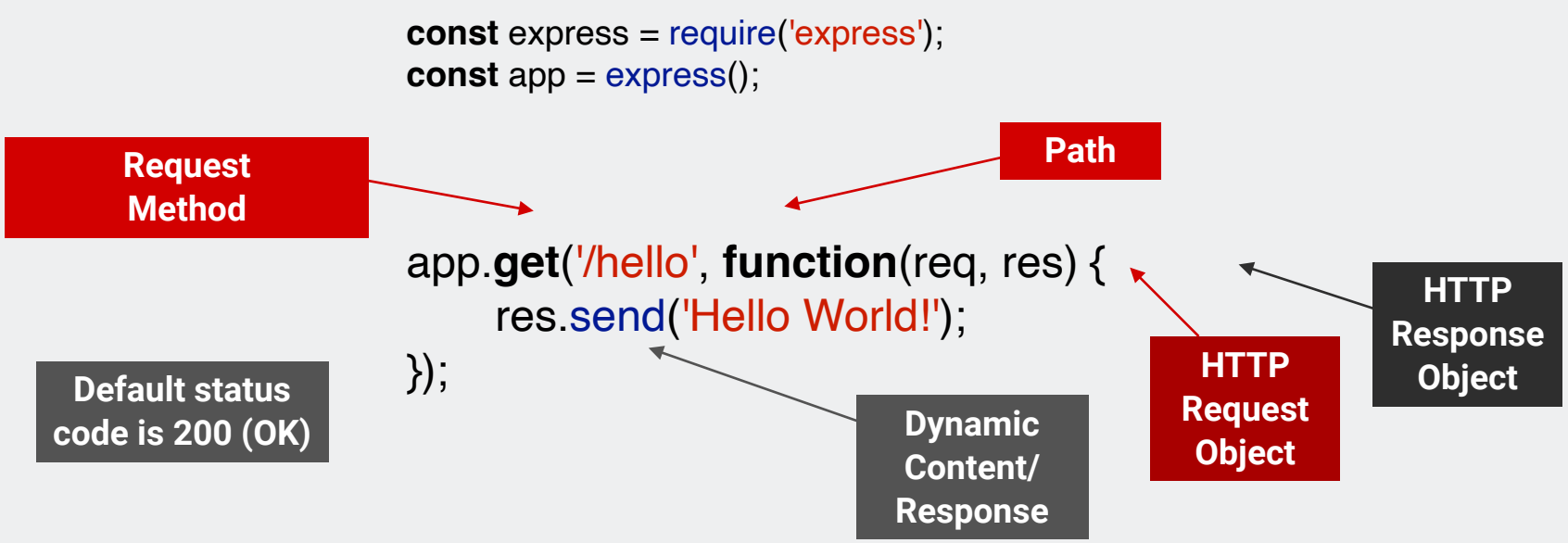
HTTP Request Headers

```
GET /path HTTP/1.1
Upgrade: websocket
Connection: Upgrade
…
```

HTTP Response Headers

```
HTTP1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
…
```

Recap: Backend Endpoints in Node.js/Express




```
const port = 3000;
app.listen(port, function() {
        console.log(`Waiting for requests on Port ${port}!`);
});
```

WebSockets Endpoints

express-ws library in Node.js/Express (light wrapper around Node ws library)

```
const express = require('express');
const app = express();
const expressWS = require('express-ws')(app);

app.ws('/create',(ws, req) => {
    ws.on('message', message => {
        console.log('Received message', message);

        ws.send('Message received' + message);
    }

    ws.on('close', _ => {
        console.log('WebSocket closed' + message);
    }
});
```

Websocket Endpoint Path

Message Event

Receive raw string messages

Send raw string messages

Socket Close Event

Broadcasting messages to all connected clients

WebSocket handles need to be stored in an established session

```
const connections = []
app.ws('/create',(ws, req) => {

    connections.push(ws);

    ws.on('message', message => {
      const actionMessage = JSON.parse(message);
      if(actionMessage.type == "join") {
        // broadcast to all stored connections
        connections.forEach(
          conn_ws => conn_ws.send('New user joined')
        );
    }
    ....
});
```

WebSockets in the Browser

```
const ws = new Websocket('localhost:3000/websocket');

ws.onopen = event => {
    ws.send('String message to the websocket server when joining');
}

// Listen for messages
ws.onmessage = event => {
    console.log(event.data);
}

ws.onclose = function(event) {
    console.log('WebSocket connection closed.');
};

document.getElementById('send').addEventListener('click', _ => {
    const message = document.getElementById('message').value;
    ws.send(message);
});
```

Websocket Endpoint Path

Connection Open Event

Send raw string messages

Ongoing Message Listening Event

Receive raw string messages

WebSocket connection closed Event

Global scope ws connection can be used in other events